Universal Transport Adaptor Configuration

Overview

Universal Transport Adaptor (hereinafter Universal TA) is a FIXEdge in-built C++ Transport Adaptor which can be used for interaction between FIXEdge and any other Transport Adaptor written on Java.

Universal TA Functionality

Universal TA is responsible for:

- Establishing and keeping connection with Java Transport Adaptor
- Messaging between FIXEdge and Java Transport Adaptor

Please see the whole interaction scheme following this link.

Interaction between Universal TA and Java TA is carried by raw FIX messages or any data (in JSON, XML, etc) wrapped in 213 tag within the FIX message with MsgType = "n". The way of interaction can be configured by means of *SmartXMLProcessing* property.

The version of the FIX protocol is specified in *Version* property.

Transport Adaptors Events

The following events are processed on the Universal TA:

EventCode	Description
INFO	Info codes
SESSION_CREATED	Session created
MESSAGE_DELIVERED	Message delivered
SESSION_DISCONNECTED	Session disconnected
SESSION_RECONNECTING	Session reconnecting
SESSION_CONNECTED	Session connected
SESSION_DESTROYED	Session was destroyed
ERROR	Error codes
ERROR_ON_CREATE_SESSION	Error on create session
ERROR_ON_PROCESS_MESSAGE	Error on process message. For sample when consumer try to read message
ERROR_ON_SEND_MESSAGE	Error on send message. For sample when producer try to send message
ERROR_ON_AUTHORIZE	Error on authorize process. For sample when we tried send/receive from topic/queue

Java Class Signature

Please see the Java class signature that can be used by any Transport Adaptor written on Java:

```
package com.epam.feta.core;

/**
 * This abstract class is needed for message communication between FE and its transport adapter (TA).
 * Each adapter will be run on a new instance of JVM.
 */
public abstract class JavaAdapter {
    public JavaAdapter() {
    }
    /**
     * FE implements this method and receives the messages from TA through it.
     * @param clientId a name of the client. It is configured in FIXEdge.properties and used in BL rules in the
capacity of sender or receiver.
     * @param message a message to FE (which has been received from TA) as byte array.
     */
    public native void onMessageCallback(String clientId, byte[] message);
    /**
     * FE implements this method and receives the events from TA through it.
     *      For example: event MESSAGE_DELIVERED when the message has been sent by TA.
     *
     * @param clientId a name of the client. It is configured in FIXEdge.properties and used in BL rules in the
capacity of sender or receiver.
     * @param eventCode the code of event from set. There is the set of events which is reserved in FE and TA.
     * @param description event description.
     * @param message this parameter is used to know what message was cause the event
     * @see com.epam.feta.core.event.EventCode
     */
    public native void onEventCallback(String clientId, long eventCode, String description, byte[] message);
    /**
     * The method is called by FE when it starts.
     * @param prefix prefix of TA's parameters. It is declared in parameter "TransportLayer.TransportAdapters"
     * @param pathToConfigFile path to FIXEdge.properties. TA parses it.
     * @throws Exception
     */
    public abstract void init(String prefix, String pathToConfigFile) throws Exception;

    /**
     * The method is called by FE when its parameters were update in runtime.
     * @param prefix prefix of TA's parameters. It is declared in parameter "TransportLayer.TransportAdapters"
     * @param config parameters and values for TA from FIXEdge.properties as string. TA parses it.
     * @throws Exception
     */
    public abstract void reInit(String prefix, String config) throws Exception;
    /**
     * The method is called by FE to send a message to TA
     * @param clientId a name of the client. It is configured in FIXEdge.properties and used in BL rules in the
capacity of sender or receiver.
     * @param message a message to TA (which has been received from FE) as byte array.
     */
    public abstract void sendMessage(String clientId, byte[] message);
    /**
     * The method is called by FE to take the parameters for monitoring tool
     * @return Name of parameter and its value(split by equals sign), this pairs will be sent to monitoring
tool as is. Parameters must be split by commas.
     */
    public abstract String getMonitoringParameters();
    /**
     * The method is called by FE when it is going to stop.
     * When this method is invoked we need to stop processing the messages to FE and process messages only from
FE.
     */
    public abstract void preDestroy();
    /**
     * The method is called by FE when it stops.
     */
    public abstract void destroy();
```

```
}

package com.epam.feta.event.client;

public enum ClientState {
    CREATED, CONNECTING, CONNECTED, DISCONNECTED, RECONNECTING, AUTHORIZATION_ERROR
}
```

Java Implementation Sample

Below is the sample of Java TA implementation:

```
package com.epam.feta.impl;

import java.io.FileReader;
import java.io.Reader;
import java.util.*;

import static com.epam.feta.core.event.EventCode.*;

public class JavaAdapterStub {
    private static final String CLIENT_NAMES = "ClientNames";
    private Set<String> clientsSet = new HashSet<>();
    private final HashMap<String, String> parameters = new HashMap<>();

    /**
     * FE implements this method and receives the messages from TA through it.
     * @param clientId a name of the client. It is configured in FIXEdge.properties and used in BL rules in the
capacity of sender or receiver.
     * @param message a message to FE (which has been received from TA) as byte array.
     */
    public native void onMessageCallback(String clientId, byte[] message);

    /**
     * FE implements this method and receives the events from TA through it.
     *      For example: event MESSAGE_DELIVERED when the message has been sent by TA.
     *
     * @param clientId a name of the client. It is configured in FIXEdge.properties and used in BL rules in the
capacity of sender or receiver.
     * @param eventCode the code of event from set. There is the set of events which is reserved in FE and TA.
     * @param description event description.
     * @param message this parameter is used to know what message was cause the event
     * @see com.epam.feta.core.event.EventCode
     */
    public native void onEventCallback(String clientId, long eventCode, String description, byte[] message);


    @Override
    public void init(String prefix, String pathToConfigFile) throws Exception {
        clientsSet.clear();
        Properties properties = new Properties();
        try(Reader reader = new FileReader(pathToConfigFile)){
            properties.load(reader);
        }
        onEventCallback(null, INFO.getCode(), properties.toString(),null);
        String values = properties.getProperty(prefix + "." + CLIENT_NAMES);
        String[] clients = values.split(",");
        for(String client: clients) {
            onEventCallback(client, SESSION_CREATED.getCode(), "Session " + client + " has been created
successful", null);
            onEventCallback(client, SESSION_CONNECTED.getCode(), "Session " + client + " has been connected
successful", null);
            clientsSet.add(client);
        }
        for(String paramName: properties.stringPropertyNames()){
            parameters.put(paramName, properties.getProperty(paramName));
```

```
        }
    }

    @Override
    public void reInit(String prefix, String config) throws Exception {

    }

    @Override
    public void sendMessage(String sessionID, byte[] message) {
        onMessageCallback(sessionID, message);
    }

    @Override
    public String getMonitoringParameters() {
        return "parameter 1 = value 1 , parameter 2=value2, parameter3= value3,parameter4=value4";
    }

    @Override
    public void preDestroy() {

    }

    @Override
    public void destroy() {
        for(String client: clientsSet) {
            onEventCallback(client, SESSION_DISCONNECTED.getCode(), "Session " + client + " has been
disconnected successful", null);
            onEventCallback(client, SESSION_DESTROYED.getCode(), "Session " + client + " has been destroyed
successful", null);
        }
    }
}
```

Universal TA Configuration

Configuration Parameters

The following configuration parameters are setup on the level of the Universal TA:

Property name	Description	Required	Default Value
TransportLayer.TransportAdapters	Comma delimited list of TA. Separate configuration section for each listed client should be specified.	Y	-
TransportLayer.JVMOptionsFile	Path to the jvmopts file that contains setting of JVM.	Y	-
TransportLayer.<Adaptor_Name>.ClientNames	Comma delimited list of TA clients. Separate configuration section for each listed client should be specified	Y	-
TransportLayer.<Adaptor_Name>.Client.<Client_Name>.Version	FIX version	N	FIX4.4
TransportLayer.<Adaptor_Name>.SmartXMLProcessing	When set to true, allows to wrap the content of any message into 213 tag of the FIX message with MsgType = "n"	N	false
TransportLayer.<Adaptor_Name>.JavaClass	Path to the java class of the specific Java TA adaptor	Y	-

All other parameters are setup on the level of specific Transport Adaptors (see JMS TA Configuration Parameters and RMQ TA Configuration Parameters f or examples)

Configuring FIXEdge.properties

Please see the sample of FIXEdge.properties file configuration with usage of Universal TA: